Phase Space Reconstruction

Caterina Ceccato, Jos Prinsen, Ethel Pruss, Anita vrins

2023-05-24

This entry will focus on Phase Space Reconstruction, which is a tool to reconstruct the multi-dimensional system dynamics from a single time series (Kantz & Schrieber, 2004). This entry will include several steps that will be fundamental for other analyses that will be presented in entries following this one.

Only part of the original dataset will be used in this entry. In particular, the data for one participant will be selected, and the phase space reconstruction for each epoch will be calculated. The aim is to investigate whether the phase space reconstruction of the EEG data of one participant will change over time. If that is the case, an analysis of possible reasons will be done.

1) Loading data and setting up

The data is loaded, and a (semi) random participant is selected.

```
library(reticulate)
require(nonlinearTseries) #nonlinearTseries, 0.2.12
require(tseriesChaos) #tseriesChaos, 0.1-13.1
require(plot3D) #plot3D, 1.4

#data is loaded and participant is selected
path_to_pklA <- "C:/Users/caty-
/Downloads/RawEEGComplexSystemsRReadyAdaptive.pkl"
RawEEGComplexSystemsRReadyAdaptive <- py_load_object(path_to_pklA)

set.seed(42)
part_n = sample(1:37, 1)
part1_ad = RawEEGComplexSystemsRReadyAdaptive[[part_n]]
```

The data for one participant contains 9 lists, corresponding to the 9 epochs, and each epoch contains 8 EEG channels, 2 columns with information about the condition and one column with the participant number.

```
#Check if the dimensions of the data correspond
print("Number of epochs: ")

## [1] "Number of epochs: "

length(part1_ad)

## [1] 9
```

```
print("Number of channels: ")

## [1] "Number of channels: "

length(part1_ad[[1]])

## [1] 11

print("1st epoch")

## [1] "1st epoch"

print(part1_ad[[1]])

##              Fz        C3        Cz  ...  AdaptiveRandom  FirstSecond
Participant
## 0    -12.877797 -3.255558 -3.537472  ...             1.0          0.0
39.0
## 1    -10.599298 -2.673650 -2.571720  ...             1.0          0.0
39.0
## 2    -10.229190 -3.916416 -2.177865  ...             1.0         -0.0
39.0
## 3    -12.108228 -5.393600 -2.425360  ...             1.0          0.0
39.0
## 4    -13.485413 -4.818951 -2.481158  ...             1.0          0.0
39.0
## ...         ...       ...       ...  ...             ...          ...
...
## 2556  -4.792877 -1.095704  0.331023  ...             1.0          0.0
39.0
## 2557  -3.542809 -1.085884  0.469226  ...             1.0          0.0
39.0
## 2558  -3.196187 -0.469547  0.481530  ...             1.0         -0.0
39.0
## 2559  -4.217521  0.402390  0.354247  ...             1.0          0.0
39.0
## 2560  -5.781861  0.896795  0.301587  ...             1.0          0.0
39.0
##
## [2561 rows x 11 columns]
```

Takens' (1981) Delay Embedding Theorem will be used to calculate the phase space reconstruction. The theorem states that the times series of one variable from a system contains information about other variables in the system. Before performing this operation, some features need to be fine-tuned.

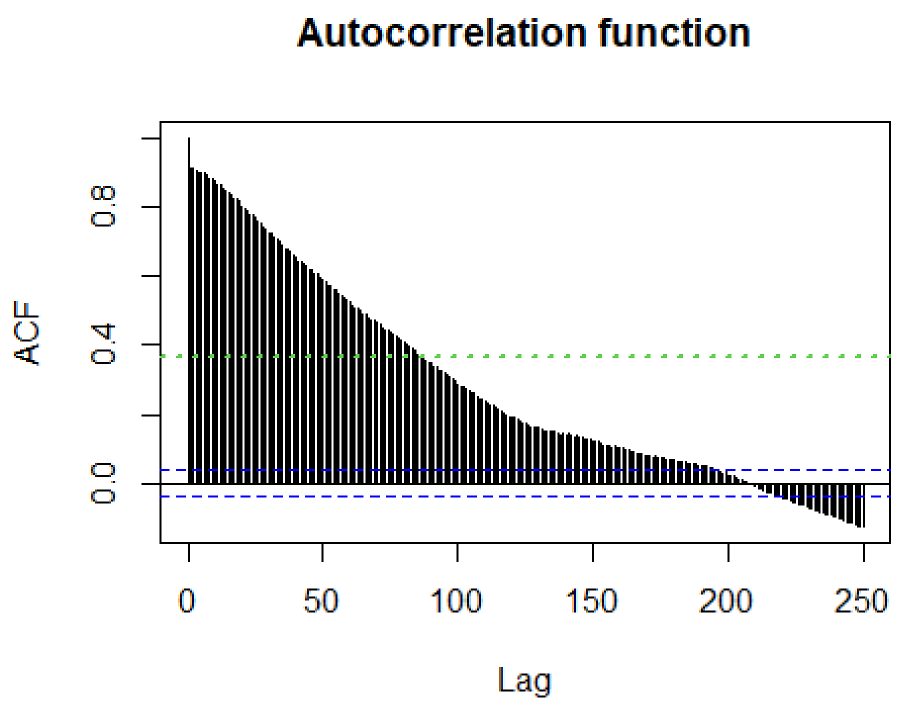
2) Calculate the Tau

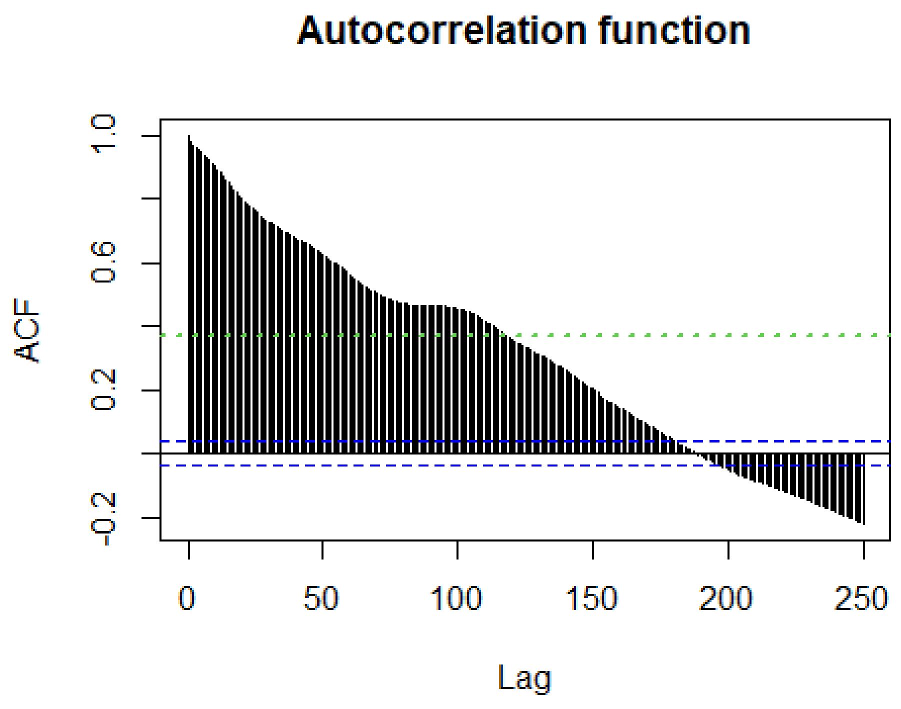
First, we need to determine the appropriate time delay (tau). The tau determines how far apart points are in the reconstructed phase space. Smaller tau values capture fine-scale dynamics, while larger tau values reveal slower processes. There are two ways to calculate it:

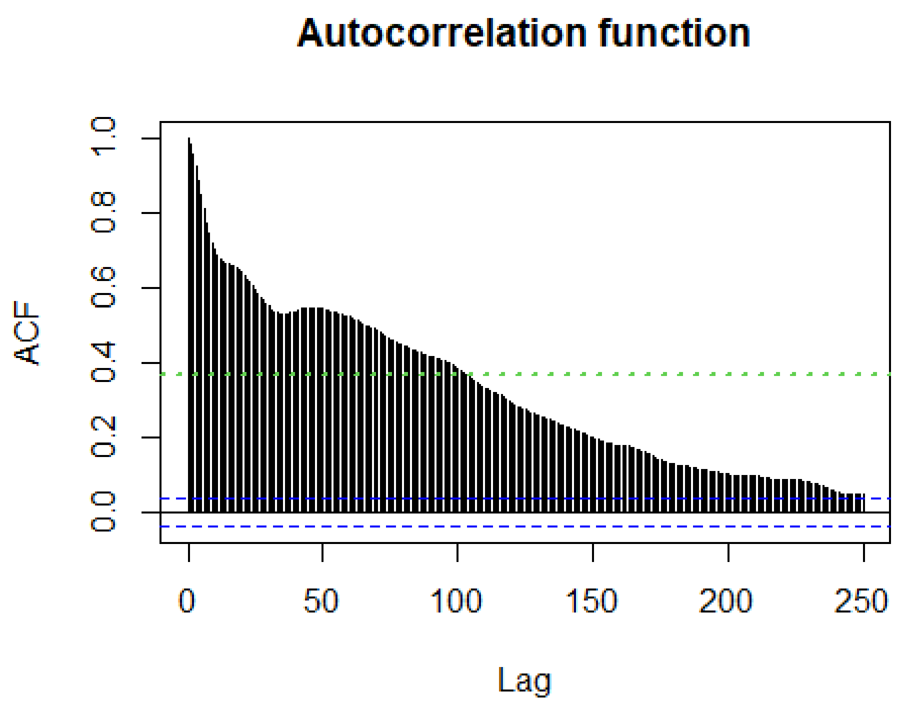
1) Looking at the autocorrelation function and looking for the first 0 crossing
2) Finding the first local minimum of the average mutual information function

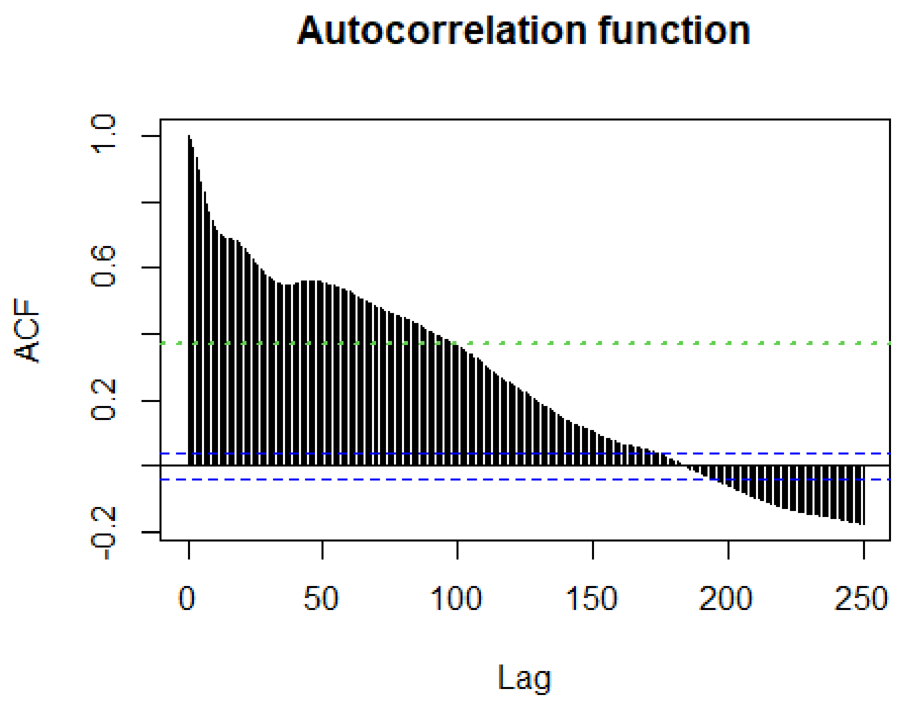
The first approach can be done as following:

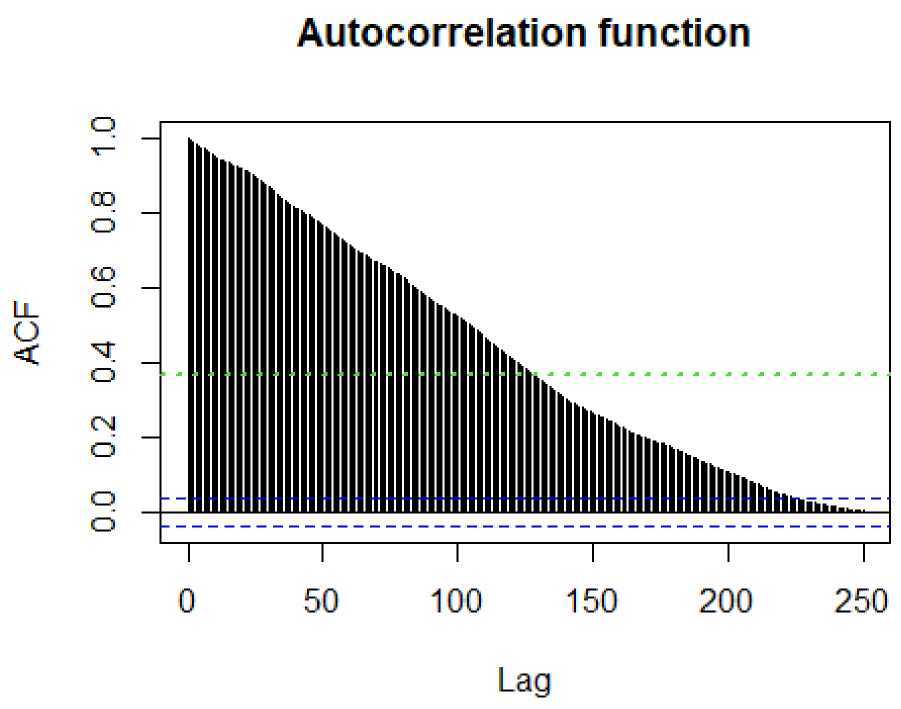
```r
channels <- c("Fz", "C3", "Cz", "C4", "Pz", "PO7", "Oz", "PO8")

acf_values <- NULL  # Initialize an empty vector to store ACF values
#loop over epochs
for (i in 1:length(part1_ad)) {
  chan_avg <- NULL
  #loop over channels to get the average EEG per epoch
  for (chan_num in channels) {
    chan <- part1_ad[[i]][chan_num]
    chan_avg <- cbind(chan_avg, chan)
  }
  chan_avg <- rowMeans(chan_avg)  # Calculate average across channels
  tau.acf <- timeLag(chan_avg, technique = "acf", lag.max = 250, do.plot = T)
# Calculate ACF
  acf_values <- c(acf_values, tau.acf)  # Append ACF values to the vector
}
```

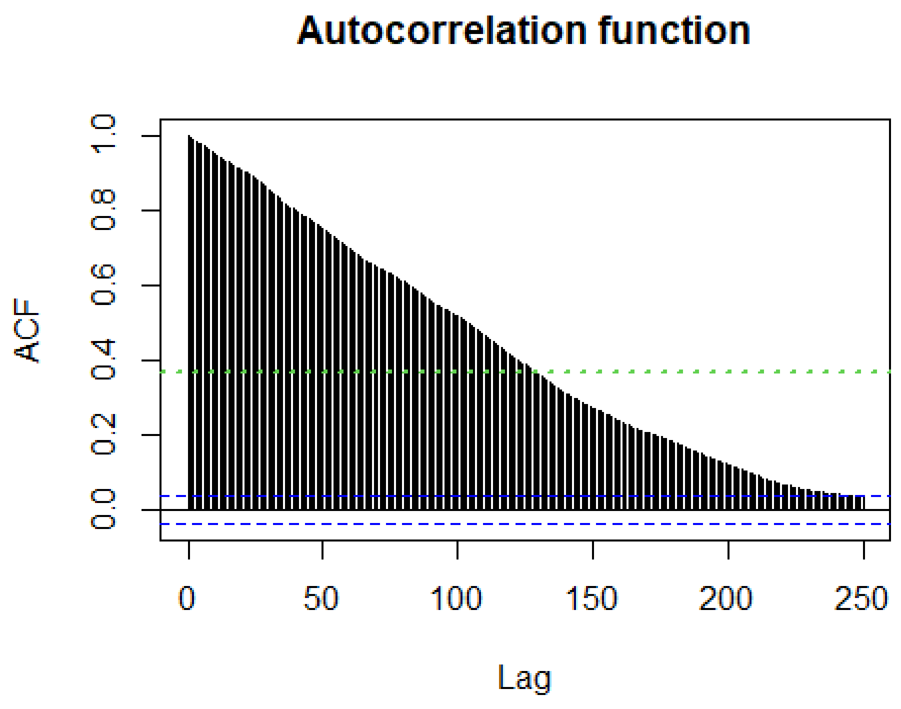

Autocorrelation function
ACF
0.0
0.4
0.8
0
50
100
150
200
250
Lag

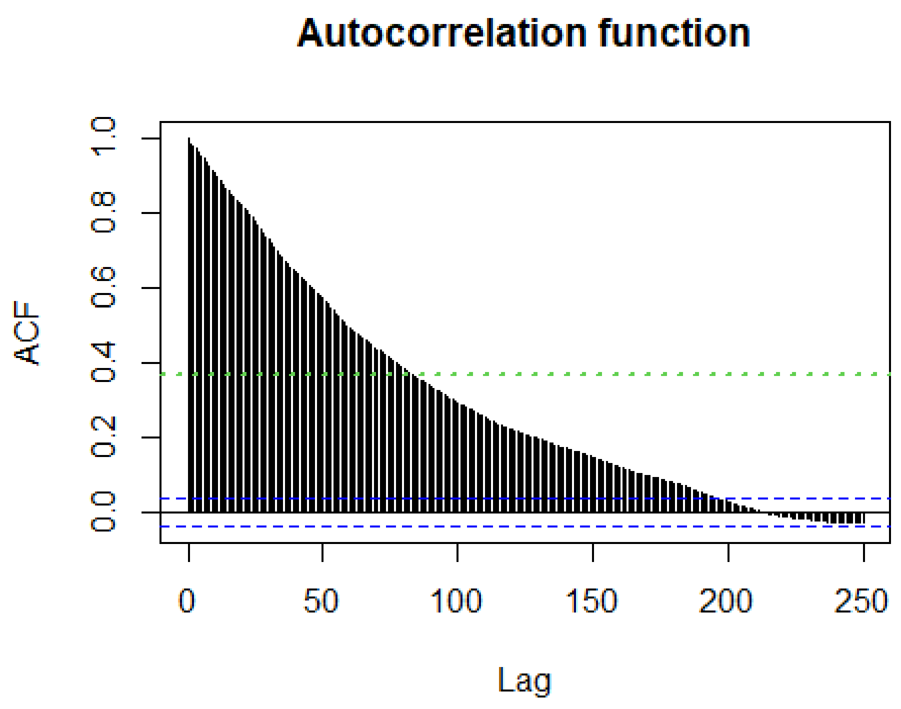

Autocorrelation function
ACF
-0.2
0.2
0.6
1.0
0
50
100
150
200
250
Lag


Autocorrelation function
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
Lag

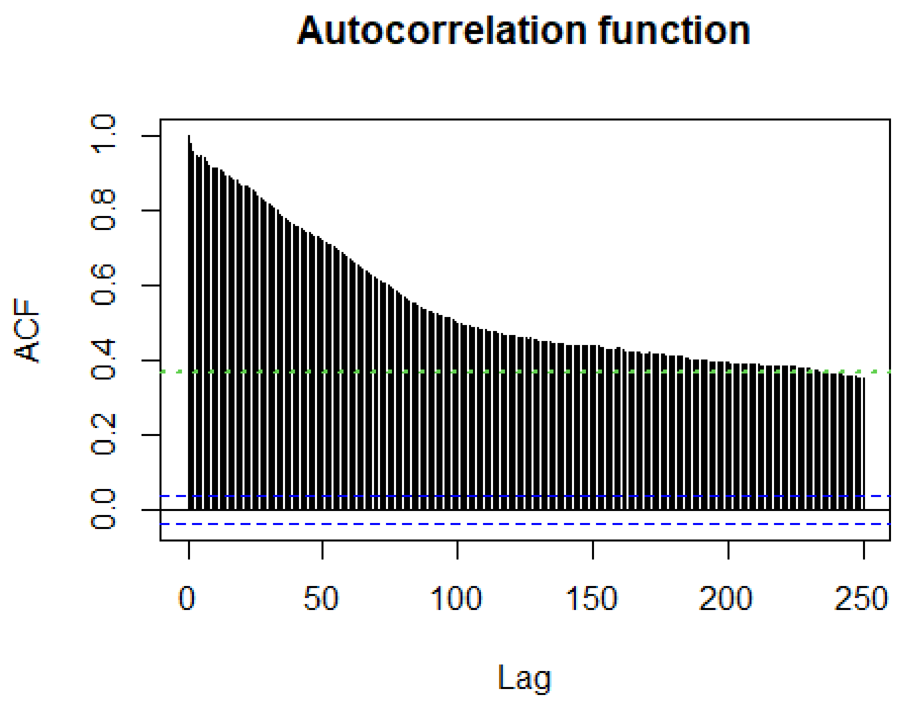

Autocorrelation function
ACF
-0.2
0.2
0.6
1.0
0
50
100
150
200
250
Lag


Autocorrelation function
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
Lag

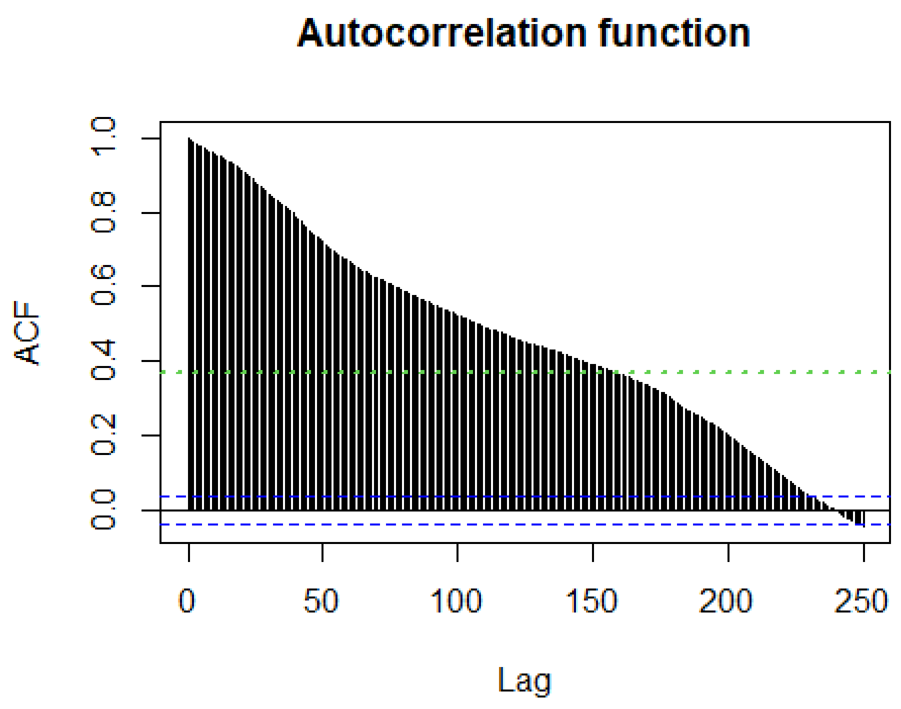

Autocorrelation function
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
Lag



Autocorrelation function
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
Lag



Autocorrelation function
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
Lag

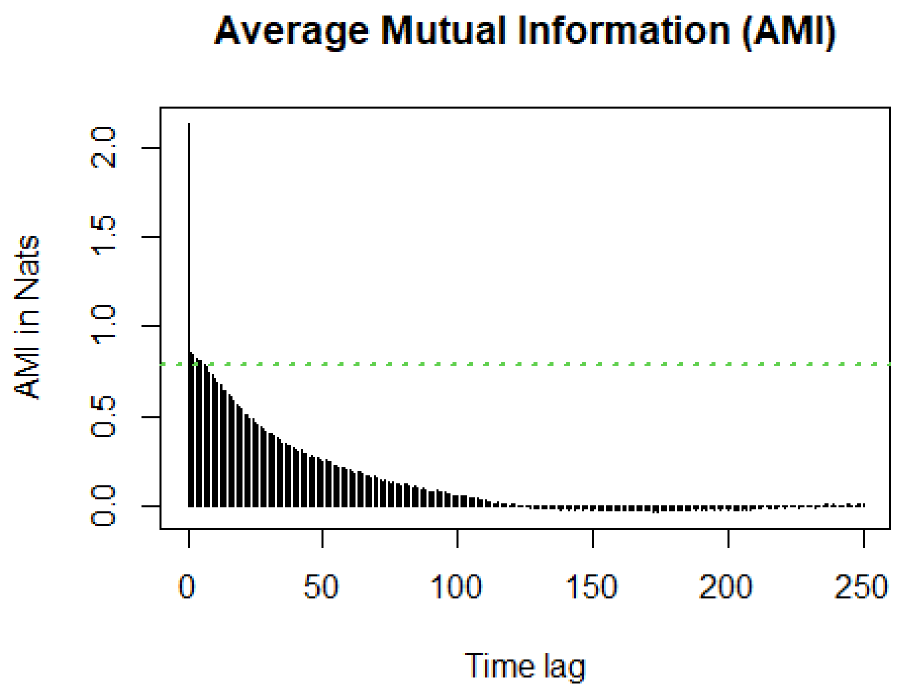


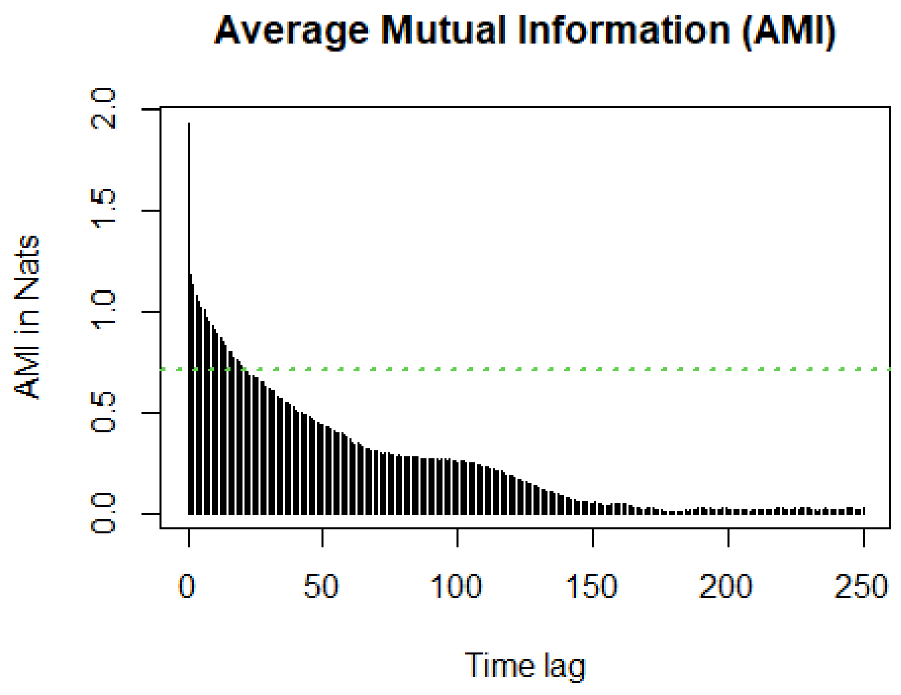
```r
#calculate the average tau
tau.acf = mean(acf_values)
print("Final (average) tau calculated with acf: ")

## [1] "Final (average) tau calculated with acf: "

print(tau.acf)

## [1] 127
```

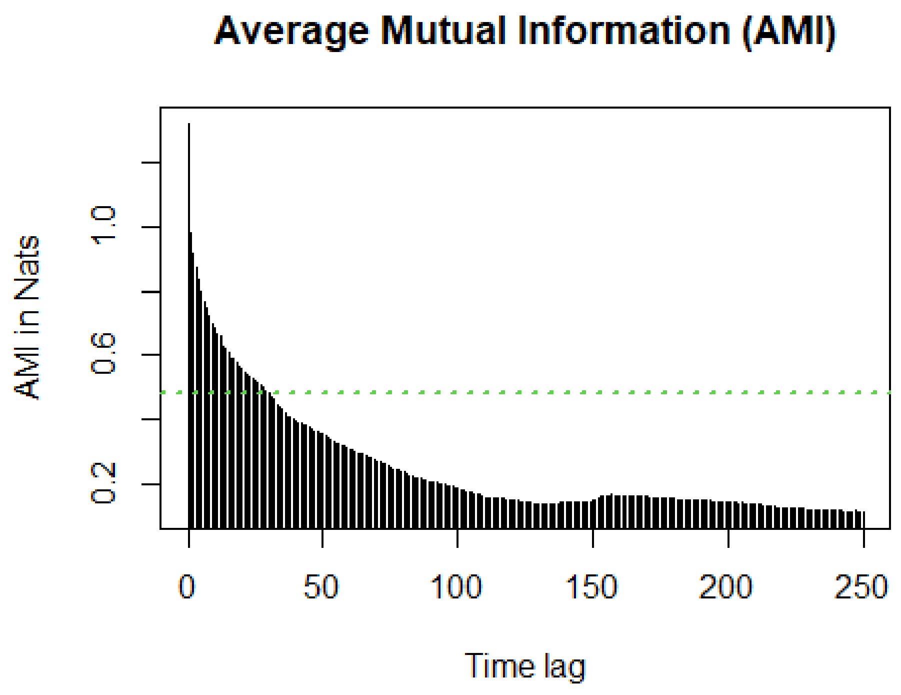
Because of the nested nature of the data, the Tau (using ACF) was calculated over each epoch (averaging over channels) and then the average of the 9 epochs was taken as definitive value. The lag.max variable was adapted to allow acf to be calculated for every epoch. This resulted in a very high tau (127), which does not seem very reasonable. In order to check this, let's explore the second approach.

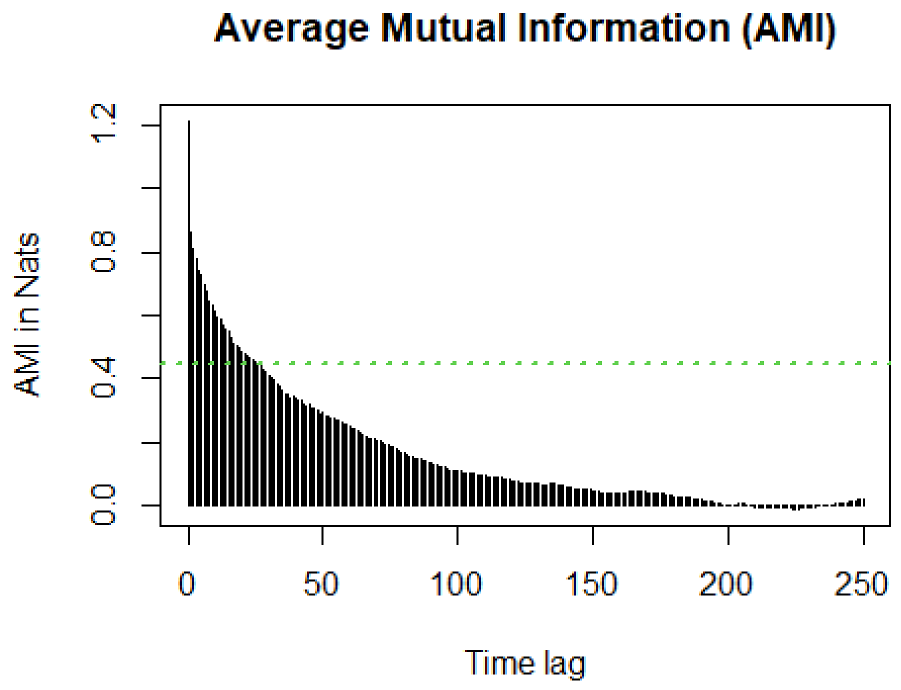
As mentioned above, this approach used Average Mutual Information. This is a measure of the amount of shared information between two variables in a dataset (Farvardin, 2003). We need to extract the first local minimum because it indicates the point at which the time series data exhibits the least redundancy and provides the most independent and informative dynamics for embedding into the phase space.

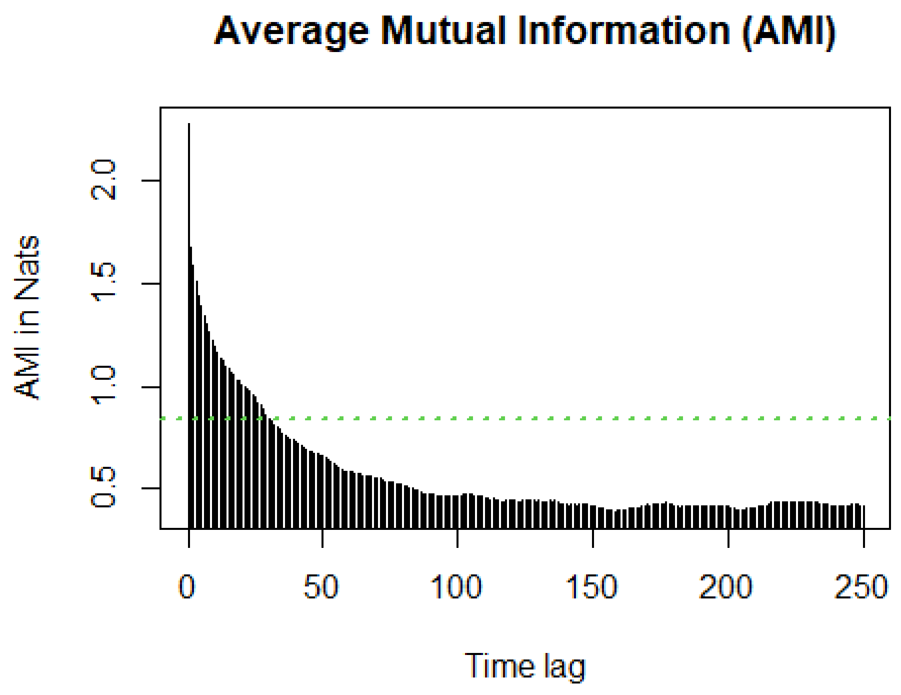
```r
channels <- c("Fz", "C3", "Cz", "C4", "Pz", "PO7", "Oz", "PO8")

ami_values <- NULL
#loop over epochs, again
for (i in 1:length(part1_ad)) {
  chan_avg <- NULL
  #loop over channels to get the average EEG over all channels
  for (chan_num in channels) {
    chan <- part1_ad[[i]][chan_num]
    chan_avg <- cbind(chan_avg, chan)
  }
  chan_avg <- rowMeans(chan_avg)  # Calculate average across channels
  #calculate the tau with ami for each epoch and store in a list
  tau.ami <- timeLag(chan_avg, technique = "ami", lag.max = 250)
  ami_values <- c(ami_values, tau.ami)
}
```


Average Mutual Information (AMI)
AMI in Nats
0.0
0.5
1.0
1.5
2.0
0
50
100
150
200
250
Time lag

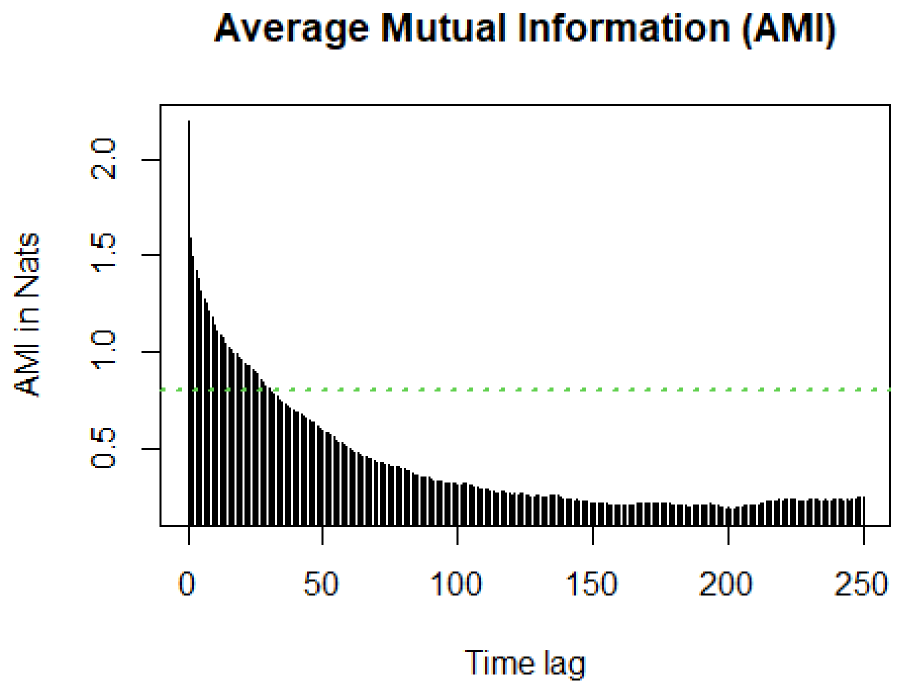

Average Mutual Information (AMI)
AMI in Nats
0.0
0.5
1.0
1.5
2.0
0
50
100
150
200
250
Time lag


Average Mutual Information (AMI)
AMI in Nats
0.2
0.6
1.0
0
50
100
150
200
250
Time lag

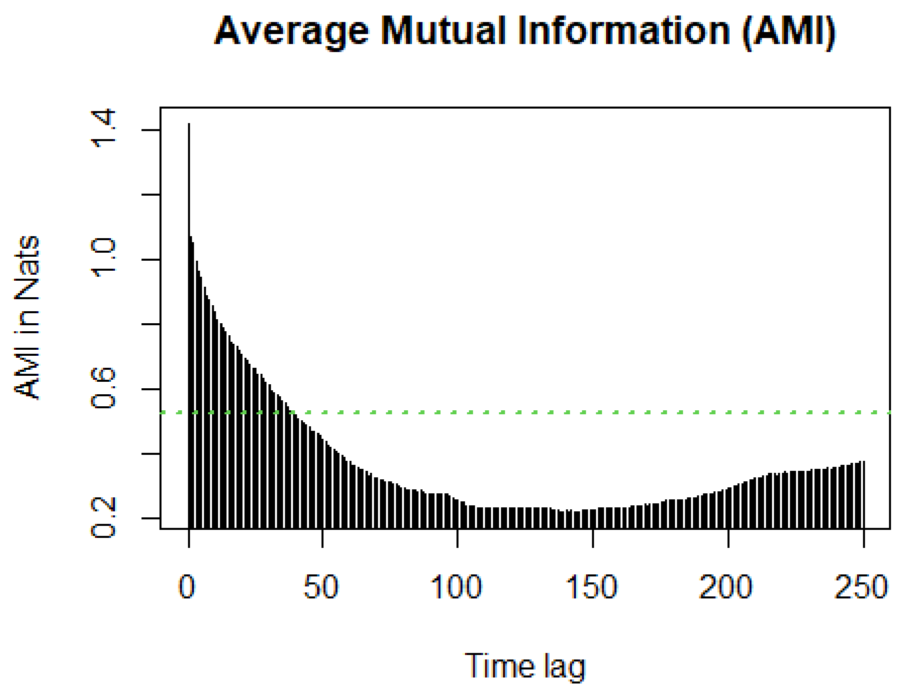

Average Mutual Information (AMI)
AMI in Nats
0.0
0.4
0.8
1.2
0
50
100
150
200
250
Time lag

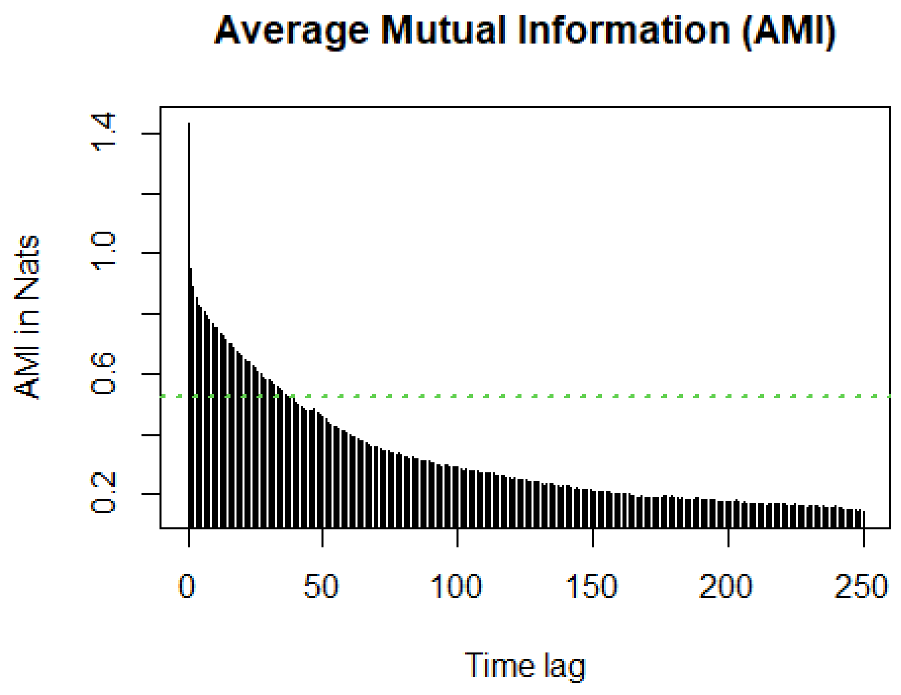


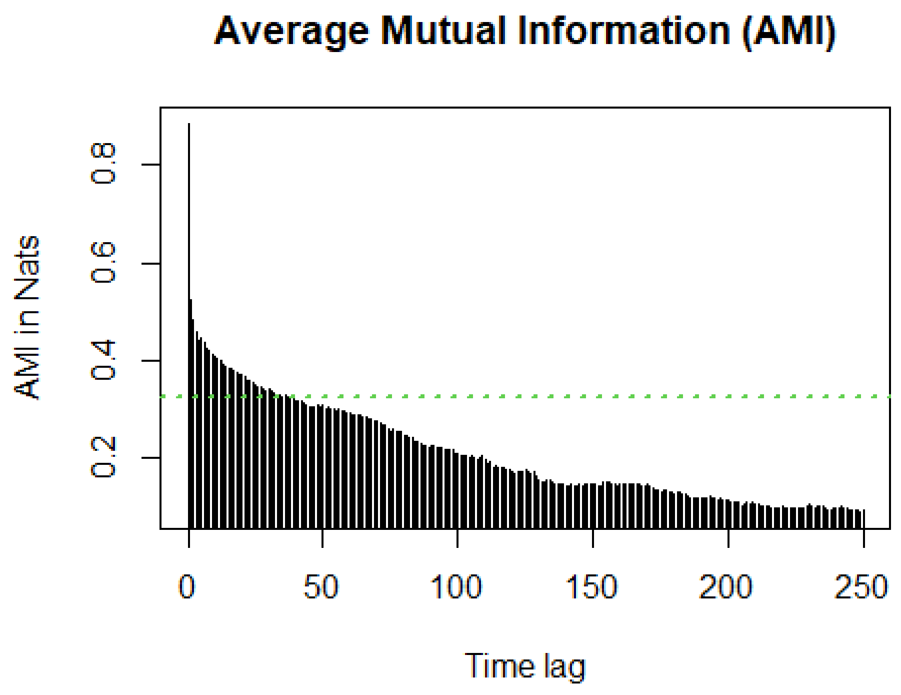
Average Mutual Information (AMI)
AMI in Nats
0.5
1.0
1.5
2.0
0
50
100
150
200
250
Time lag



Average Mutual Information (AMI)
AMI in Nats
0.5
1.0
1.5
2.0
0
50
100
150
200
250
Time lag


Average Mutual Information (AMI)
AMI in Nats
0.2
0.6
1.0
1.4
0
50
100
150
200
250
Time lag


Average Mutual Information (AMI)
AMI in Nats
0.2
0.6
1.0
1.4
0
50
100
150
200
250
Time lag



```
#average over the tau values
tau.ami = round(mean(ami_values))
print("Final (average) tau calculated with ami: ")

## [1] "Final (average) tau calculated with ami: "

print(tau.ami)

## [1] 29
```

The value of tau calculated through AMI (29) is very different than the one calculated with ACF. Considering the non-linear nature of EEG activity, it is possible that ACF cannot be properly applied to the data. AMI, on the other hand, is non-linear and might fit the data better. Moreover, as mentioned earlier, the value that was extracted from ACF was quite high, and that might lead to more missing data. For these reasons, AMI will be used for the rest of the analysis.

3) Calculate the embedding dimension

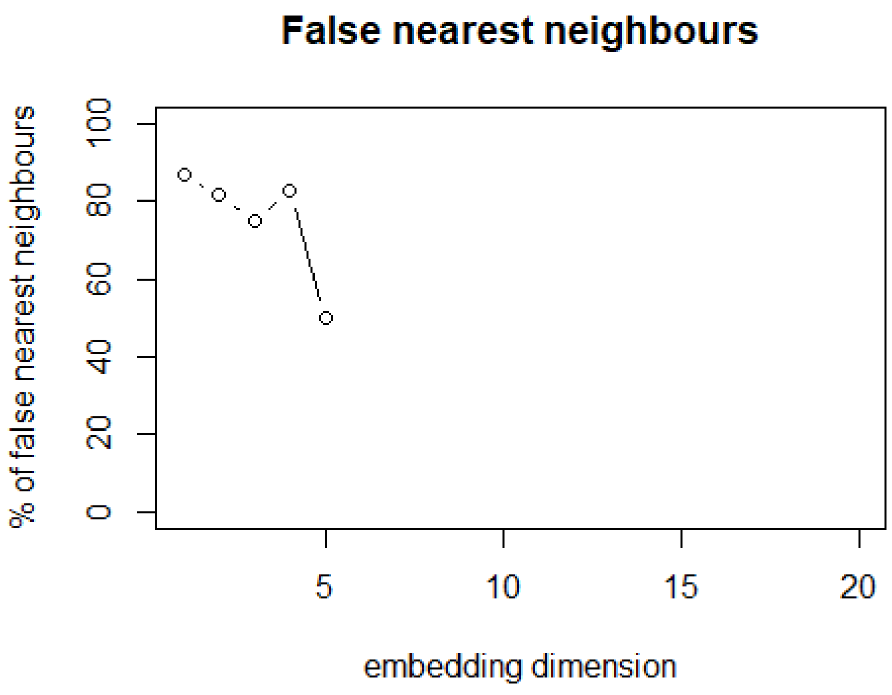
The next step is to calculate the embedding dimension, which is the number of dimensions that will be used to represent the reconstructed phase space. Basically, it determines what the dimension of the space will be. Also in this case, two techniques can be used:

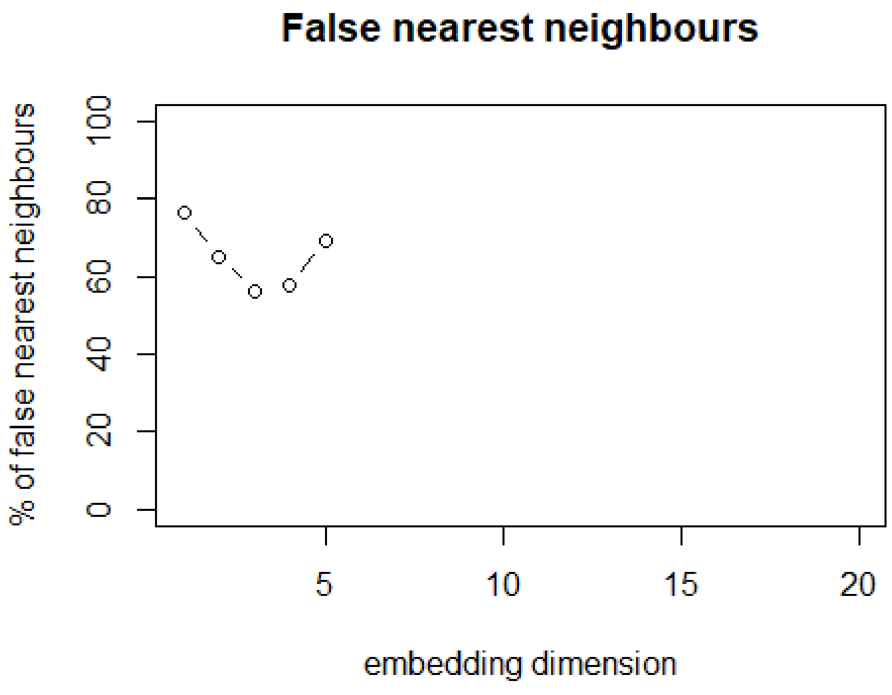
1) False nearest neighbors methodology (Kennel et al., 1992). This method aims at detecting dimensions that are "false neighbors", meaning that they are not really necessary to explain the data.

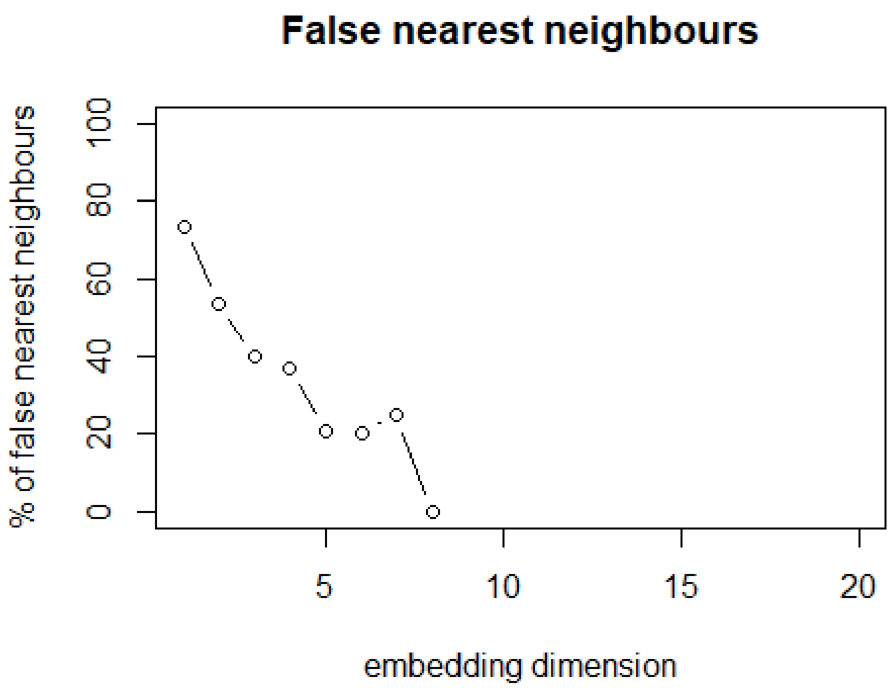
2) Cao's (1997) algorithm. This is a heuristic approach, based on AMI to determine the optimal embedding dimension.

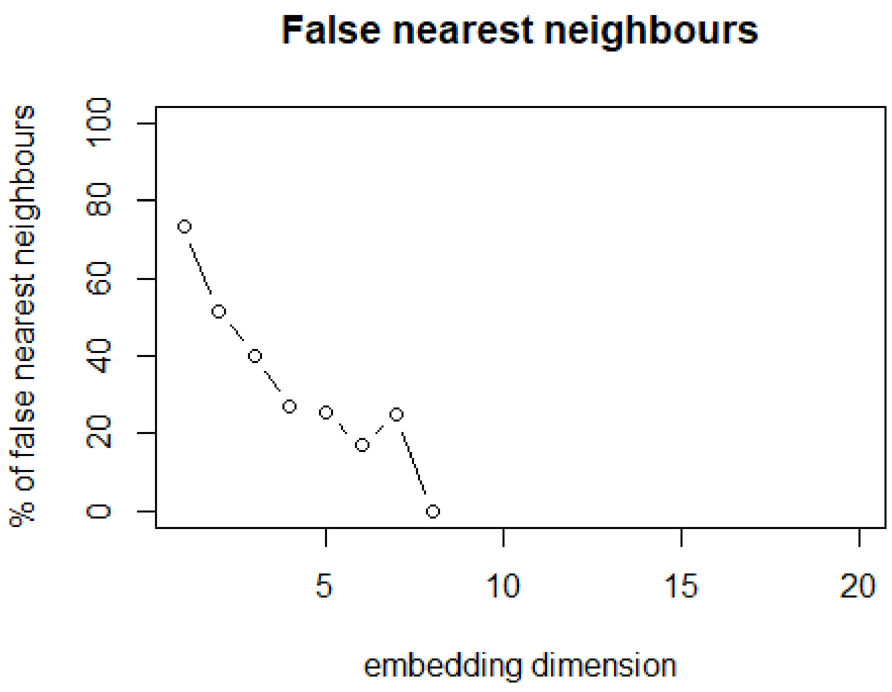
We will first demonstrate the approach using FNN:

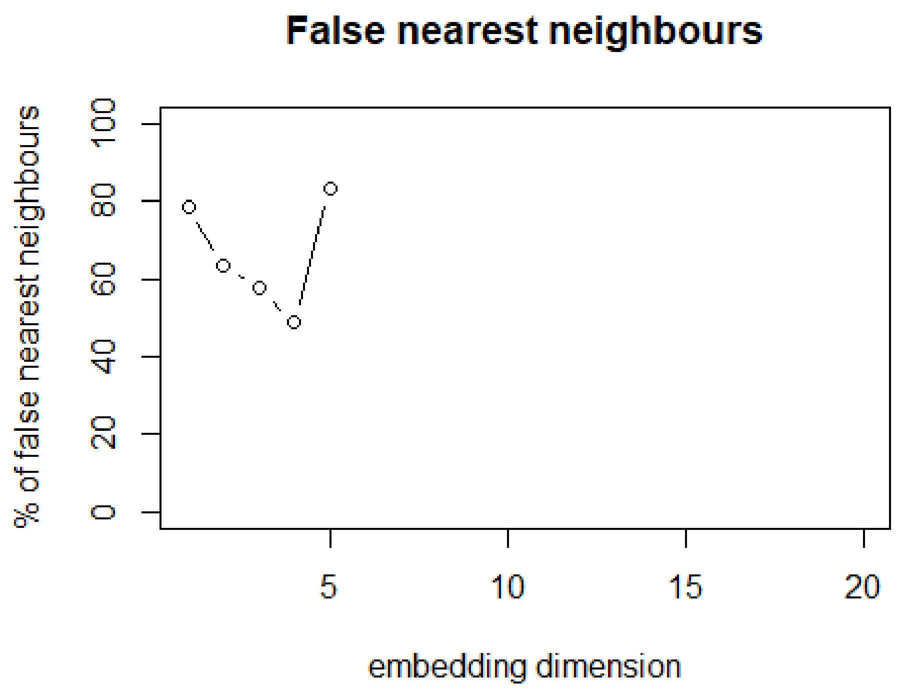
```r
fnn_values <- NULL
#loop over epochs
for (i in 1:length(part1_ad)) {
  chan_avg <- NULL
  #loop over channels to get the average EEG over channels per epoch
  for (chan_num in channels) {
    chan <- part1_ad[[i]][chan_num]
    chan_avg <- cbind(chan_avg, chan)
  }
  chan_avg <- rowMeans(chan_avg)  # Calculate average across channels
  #calculate and plot the FNN
  fnn.out = false.nearest(chan_avg, m = 20, d = tau.ami, t = 50, eps =
sd(chan_avg/10 ))
  plot(fnn.out)
  #get the embedding dimension calculated by FNN and store in a list with all
the value for all epochs
  fraction_values <- fnn.out["fraction", ]
  fraction_values <- as.numeric(fraction_values)

  fnn_values <- c(fnn_values, which.min(fraction_values >= 0.1))
}
```

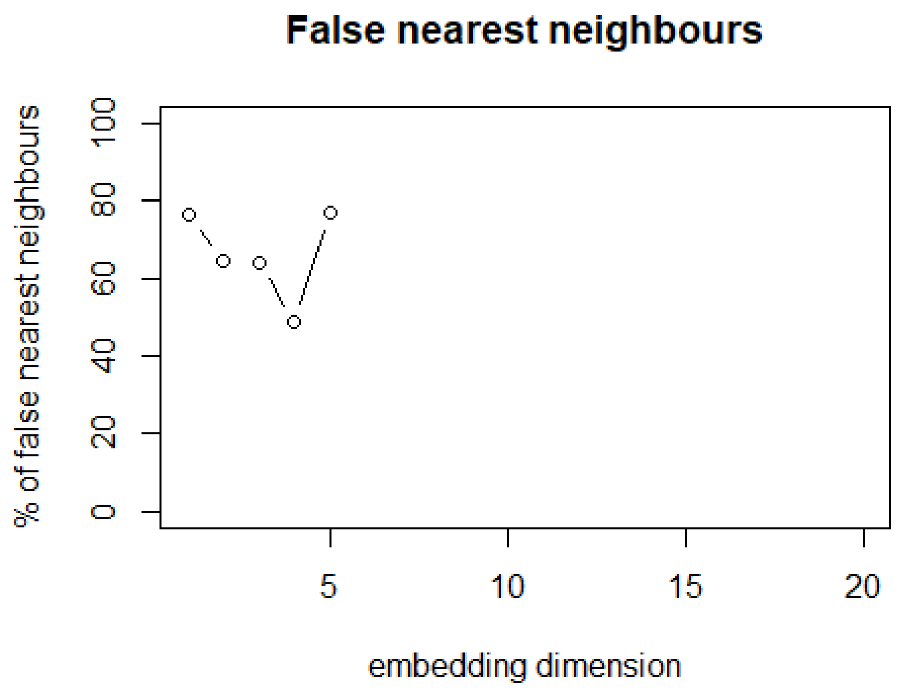

False nearest neighbours
% of false nearest neighbours
embedding dimension
0
20
40
60
80
100
5
10
15
20

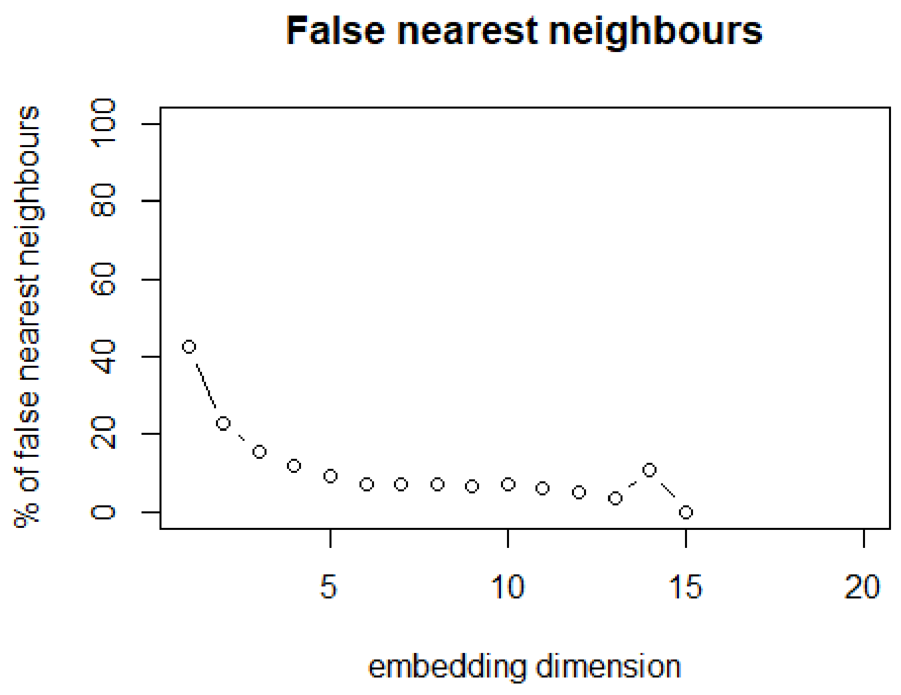

False nearest neighbours
% of false nearest neighbours
embedding dimension
0
20
40
60
80
100
5
10
15
20

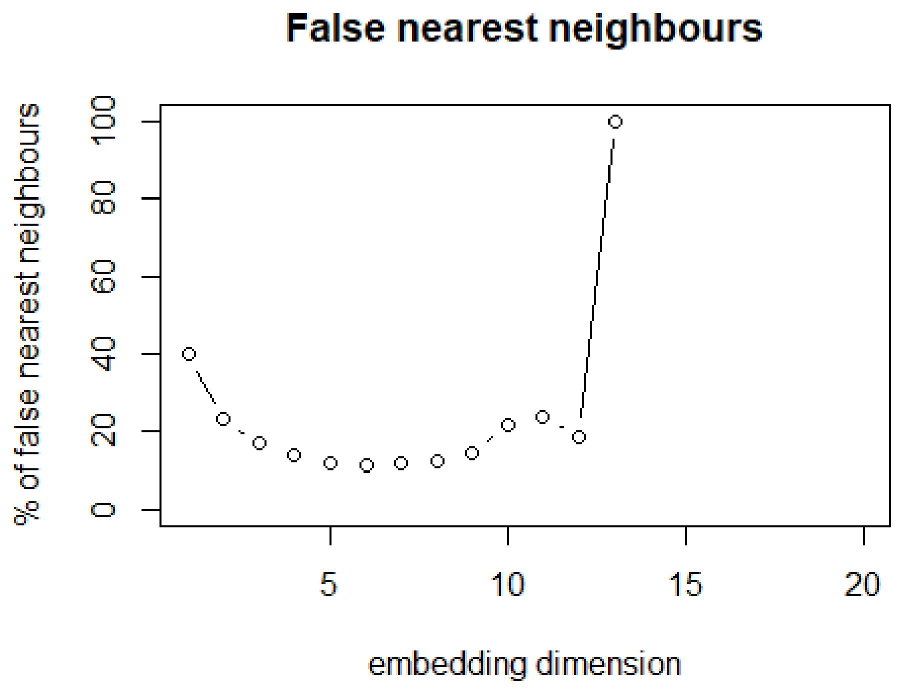

False nearest neighbours
% of false nearest neighbours
0
20
40
60
80
100
5
10
15
20
embedding dimension

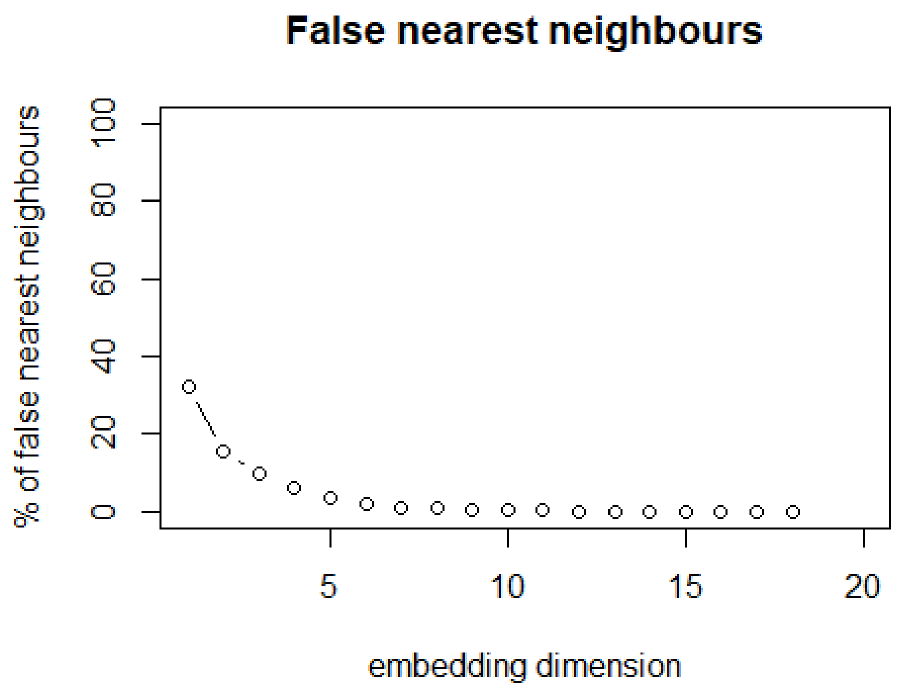

False nearest neighbours
% of false nearest neighbours
0
20
40
60
80
100
5
10
15
20
embedding dimension


False nearest neighbours
% of false nearest neighbours
0
20
40
60
80
100
5
10
15
20
embedding dimension


False nearest neighbours
% of false nearest neighbours
0
20
40
60
80
100
5
10
15
20
embedding dimension


False nearest neighbours
% of false nearest neighbours
0
20
40
60
80
100
5
10
15
20
embedding dimension

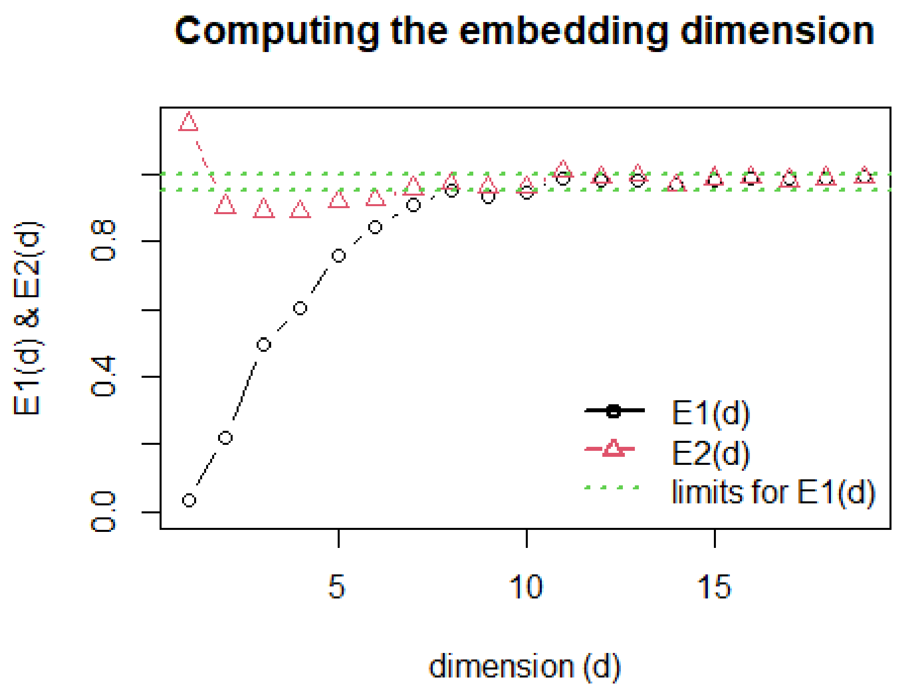

False nearest neighbours
% of false nearest neighbours
0
20
40
60
80
100
5
10
15
20
embedding dimension

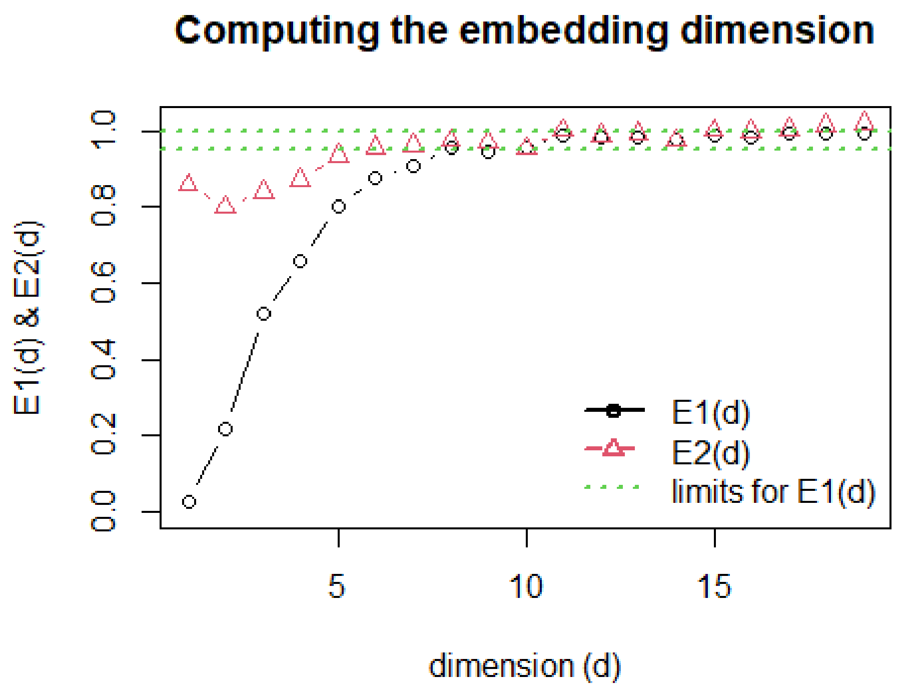


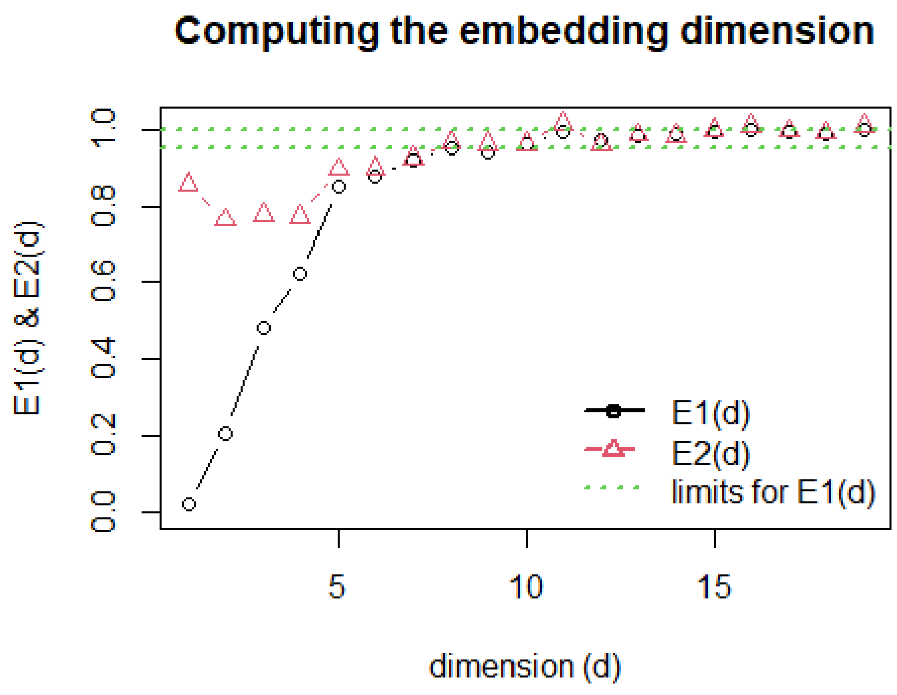
```
#take the average embedding dimension over epochs
fnn.out = mean(fnn_values)
print("Final (average) embedding dimension calculated with FNN: ")

## [1] "Final (average) embedding dimension calculated with FNN: "

print(fnn.out)

## [1] 3.333333
```

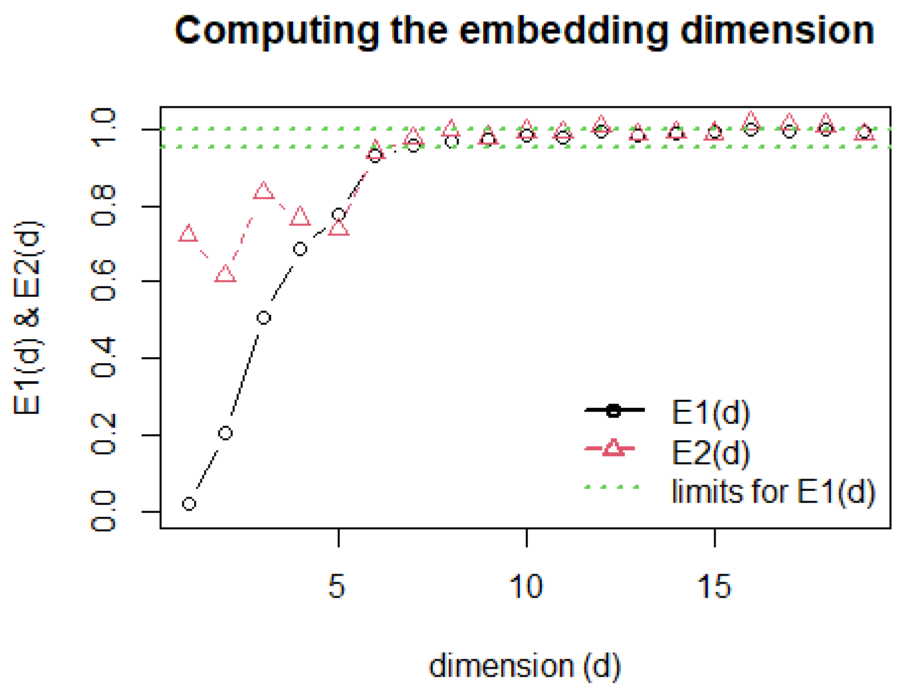
The graphs show the percentage of False Nearest Neighbour found at different dimensions. The lower the percentage, the more sure the embedding dimension is. In some of the plots generated the line does not seem to go past the first 7 embedding dimensions. After trying with different max embedding dimensions, the results seems to be consistent with what can be seen here above. This could be due to several different reasons, but first it could be useful to compare these results to the results of Cao's algorithm.

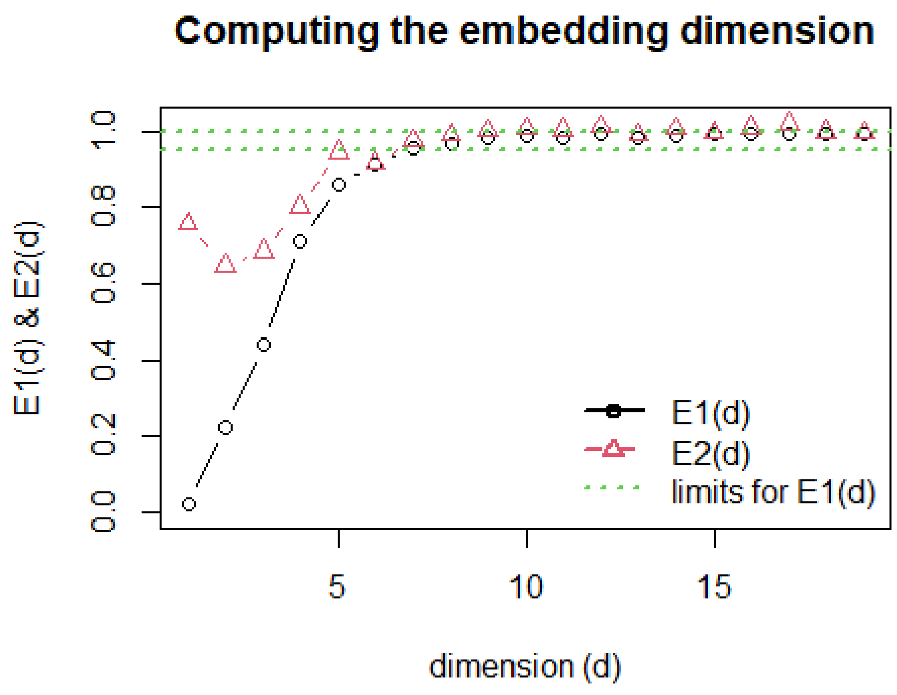
```
cao_values <- NULL
#same procedure as above
for (i in 1:length(part1_ad)) {
  chan_avg <- NULL
  for (chan_num in channels) {
    chan <- part1_ad[[i]][chan_num]
    chan_avg <- cbind(chan_avg, chan)
  }
  chan_avg <- rowMeans(chan_avg)
  #calculate embedding dimension and save in a list
  emb.dim = estimateEmbeddingDim(chan_avg, time.lag = tau.ami,
max.embedding.dim = 20)
  cao_values <- c(cao_values, emb.dim)
}
```



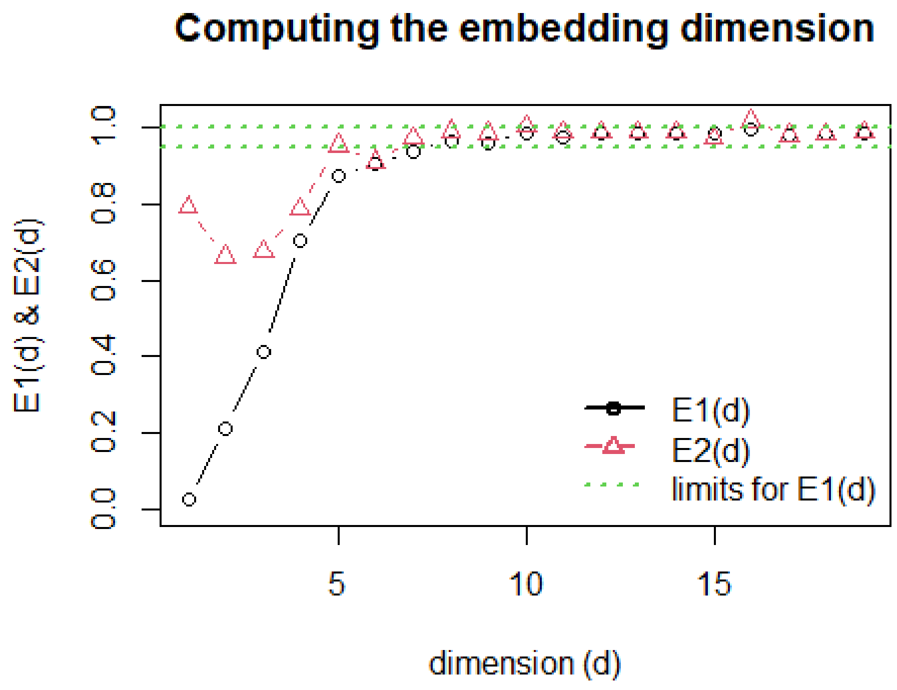



Computing the embedding dimension
E1(d) & E2(d)
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
dimension (d)
E1(d)
E2(d)
limits for E1(d)

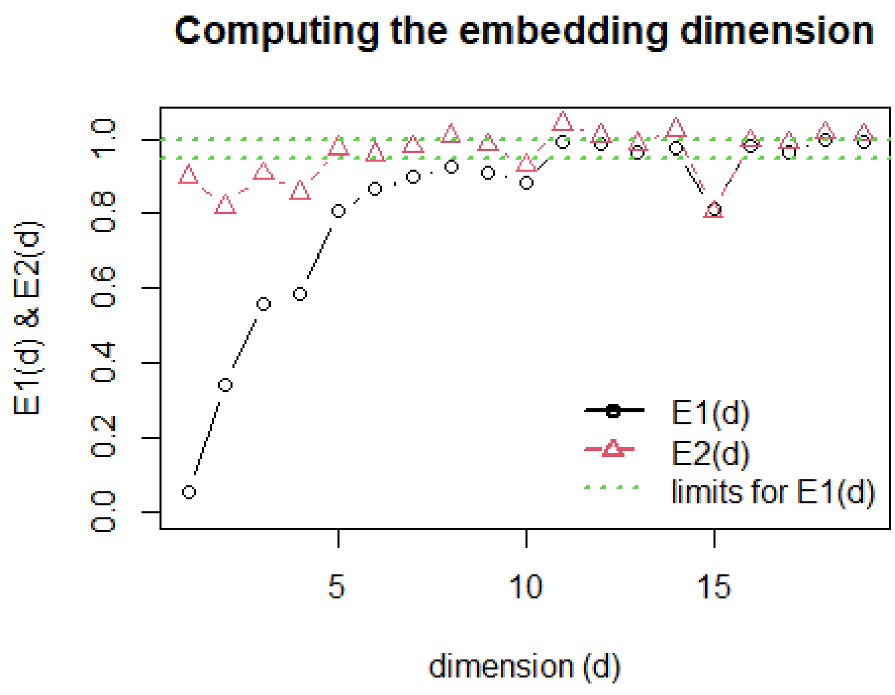

Computing the embedding dimension
E1(d) & E2(d)
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
dimension (d)
E1(d)
E2(d)
limits for E1(d)



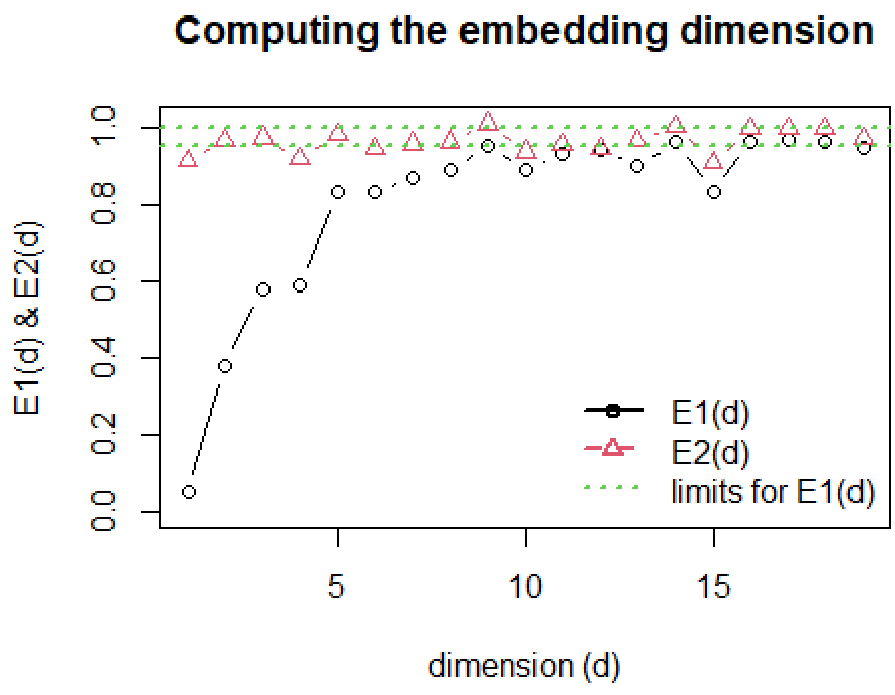
Computing the embedding dimension
E1(d) & E2(d)
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
dimension (d)
E1(d)
E2(d)
limits for E1(d)



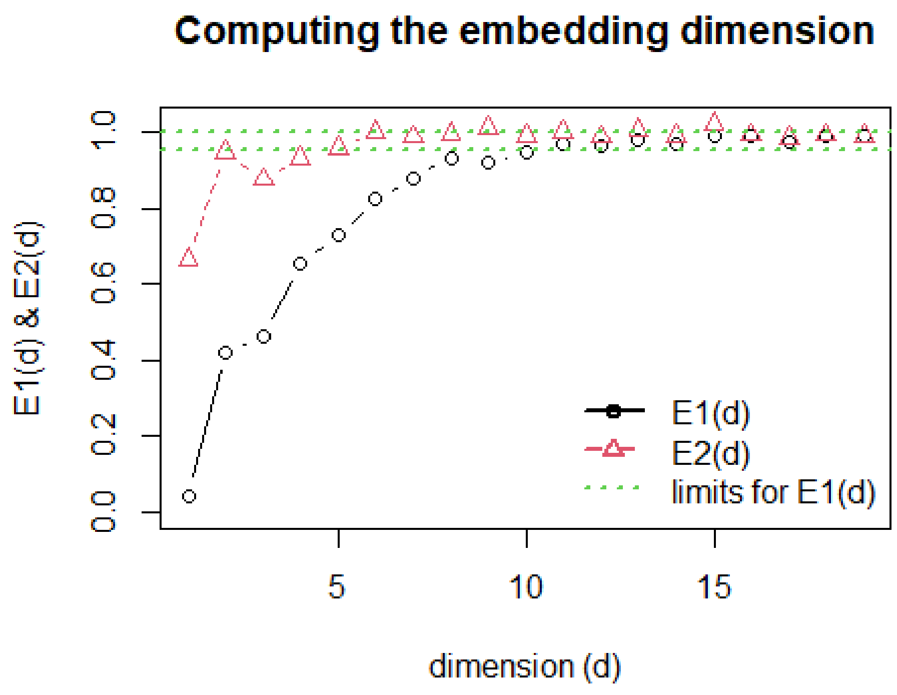
Computing the embedding dimension
E1(d) & E2(d)
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
dimension (d)
E1(d)
E2(d)
limits for E1(d)



Computing the embedding dimension
E1(d) & E2(d)
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
dimension (d)
E1(d)
E2(d)
limits for E1(d)



Computing the embedding dimension
E1(d) & E2(d)
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
dimension (d)
E1(d)
E2(d)
limits for E1(d)



```
# get the average embedding dimension
emb.dim = round(mean(cao_values))
print("Final (average) embedding dimension calculated with Cao's algorithm:
")

## [1] "Final (average) embedding dimension calculated with Cao's algorithm:
"

print(emb.dim)

## [1] 9
```

The results from Cao's algorithm seem to work better, in fact it was possible to identify an average embedding dimension that seems consistent, and the graphs are clear. In order to interpret the graph, it is important to look at E1(d) and look at when that goes above the dotted green line (is approximately 1).

The fact that the results from FNN were so inconsistent, whereas the results from Cao's algorithm are more clear, shows that the issues with FNN might be due to the data used. First of all, Cao's algorithm is more robust to noise, so it could be that the data is too noisy for FNN to be able to handle it. A second reason could be that EEG is usually very complex and high-dimensional, and Cao's algorithm can handle these characteristics better than FNN.

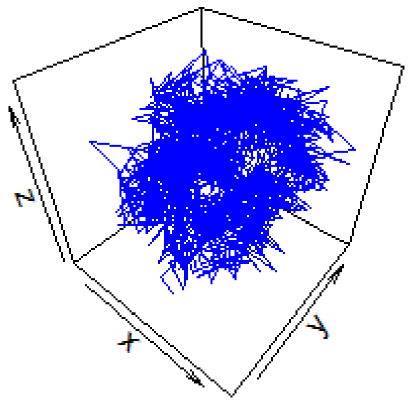
4) Phase Space reconstruction

Now that tau and embedding dimensions have been calculated, it is time to do the phase space reconstruction

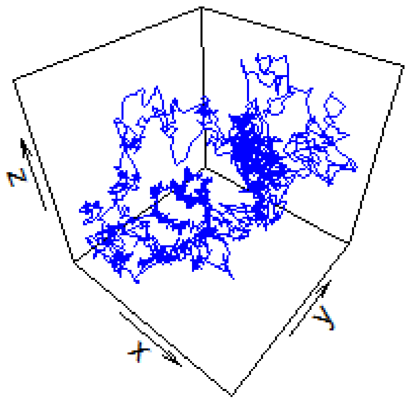
```
print("Tau: ")

## [1] "Tau: "

print(tau.ami)

## [1] 29

print("Embedding dimension: ")

## [1] "Embedding dimension: "

print(emb.dim)

## [1] 9

takens_values <- NULL
#same loop as before
for (i in 1:length(part1_ad)) {
  chan_avg <- NULL
  for (chan_num in channels) {
    chan <- part1_ad[[i]][chan_num]
    chan_avg <- cbind(chan_avg, chan)
  }
  chan_avg <- rowMeans(chan_avg)
  #calculate takens algorithm for each epoch and store in a list
  chan_avg.takens <- buildTakens(chan_avg, emb.dim,tau.ami)

  takens_values <- c(takens_values, list(chan_avg.takens))
}
```

The code above performed phase space reconstruction for each epoch, all using the same tau (29) and embedding dimension (9), for consistency. The phase space reconstructions for the separate epochs are stored in a variable, so they can all be accessed and plotted. The next step is to select some epochs to compare. Moreover, since it is not possible to plot something in 9 dimensions, 3 dimensions need to be chosen.

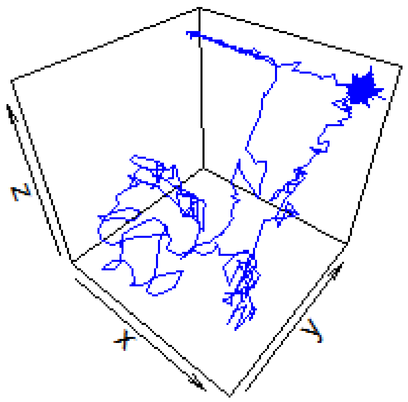
```
epoch1 = takens_values[[1]]
epoch5 = takens_values[[5]]
epoch9 = takens_values[[9]]

#plot the phase space reconstructioin in 3D for different dimensions of
epochs 1,5 and 9

lines3D(epoch1[,1],epoch1[,5],epoch1[,8], t="l", col="blue", asp=1)
```



```
lines3D(epoch5[,5],epoch5[,7],epoch5[,9], t="l", col="blue", asp=1)
```



```
lines3D(epoch9[,3],epoch9[,7],epoch9[,6], t="l", col="blue", asp=1)
```



After checking a few different epochs, we chose to use epochs 1, 5, and 9 to demonstrate the change over time. In fact, the plot showed that the signal seemed to be unraveling to create a more clear shape as the epochs increased. This did not change by changing the selection of dimensions that were plotted. This result seems quite interesting as it shows a clear change in the phase space reconstruction over time. This could for example mean that the system is gradually stabilizing or reaching a steady state. As time progresses, the system dynamics become more predictable and exhibit less variability, resulting in a more discernible pattern in the phase space reconstruction. It could also be a result of emerging dynamics, that end up creating more distinct patterns. However, it is also possible that it is simply due to a bigger amount the noise in the first epochs. It is difficult to discern what exactly the motivation could be, but the results definitely justify further analysis, either with other methods, or exploring the same method with different participants/condition.

5) Conclusion

This entry aimed at exploring the concept of phase space reconstruction and served as the basis for analyses that will be explored in future modules. In particular, it focused on the concept of Tau and embedding dimensions, which are fundamental for many other analyses. The methods to calculate these features were presented, and a brief analysis of how to choose which one to use, based on our EEG data, was done. Furthermore, the phase space reconstruction of EEG data over different epochs was calculated, which showed an evolution, from the first epoch, with a very chaotic plot, to a more and more organized plot in the fifth and the last epochs. This could provide valuable insight on the learning process, and it is an interesting find that can be further validated by future analyses. Different methods could be applied to the same data to investigate the phenomenon from different perspectives, or the same analysis could be done by looking, for example, at single channels over the whole interaction, or comparing between participants or conditions.

References

Antonio, Narzo FD (2019). *tseriesChaos: Analysis of Nonlinear Time Series*. R package version 0.1-13.1, https://CRAN.R-project.org/package=tseriesChaos.

Cao, L. (1997). Practical method for determining the minimum embedding dimension of a scalar time series. Physica D: Nonlinear Phenomena, 110(1-2), 43-50.

Farvardin, N. (2003). Source coding, theory and applications. Encyclopedia of Physical Science and Technology, 377–395. https://doi.org/10.1016/b0-12-227410-5/00714-6

Garcia C (2022). *nonlinearTseries: Nonlinear Time Series Analysis*. R package version 0.2.12, https://CRAN.R-project.org/package=nonlinearTseries.

Kantz & Schrieber (2004). Nonlinear Time Series Analysis. Ch. 3 Phase space methods.

Kennel, M. B., Brown, R., & Abarbanel, H. D. I. (1992). Determining embedding dimension for phase-space reconstruction using a geometrical construction. Physical Review A, 45(6), 3403–3411.

Soetaert K (2021). *plot3D: Plotting Multi-Dimensional Data*. R package version 1.4, https://CRAN.R-project.org/package=plot3D.

Takens, F. (1981). Detecting strange attractors in turbulence. In D. A. Rand & L.-S. Young (Eds.), Dynamical systems and turbulence, Warwick 1980 (pp. 366–381). Springer Berlin Heidelberg.